The ANGELINA Videogame Design System, Part I

Michael Cook, Simon Colton, and Jeremy Gow
http://ccg.doc.gold.ac.uk

***Abstract*—Automatically generating content for videogames has long been a staple of game development and the focus of much successful research. Such forays into content generation usually concern themselves with producing a specific game component, such as a level design. This has proven a rich and challenging area of research, but in focusing on creating separate parts of a larger game, we miss out on the most challenging and interesting aspects of game development. By expanding our scope to the automated design of entire games, we can investigate the relationship between the different creative tasks undertaken in game development, tackle the higher-level creative challenges of game design, and ultimately build systems capable of much greater novelty, surprise and quality in their output.**

This paper, the first in a series of two, describes two case studies in automating game design, proposing cooperative coevolution as a useful technique to use within systems that automate this process. We show how this technique allows essentially separate content generators to produce content that complements each other. We also describe systems that have used this to design games with subtle emergent effects. After introducing the technique and its technical basis in this paper, in the second paper in the series we discuss higher level issues in automated game design, such as potential overlap with computational creativity and the issue of evaluation.

***Index Terms*—procedural content generation, automated game design, computational creativity**

I. Introduction

PROCEDURAL generation of content for videogames is a crucial area of research and development for the games industry as it stands today. The need for highly diverse and quality content in large quantities has continued to pressure game developers at all scales. At the same time, procedurality is having an increasingly important influence on the culture of videogames, and those games which explore the ideas and themes of procedural generation in innovative ways are well received as a result. While much work has been done applying procedural content generation (PCG) techniques to individual aspects of game design, to the best of our knowledge there has been no major concerted effort to investigate the possibility of a system which procedurally generates many or even all aspects of a game's design autonomously. This is mostly likely due to the way in which PCG is typically employed in modern game development: as a tool to solve a specific problem, or a feature within a large creative vision that stems from a person.

Developing a system that can act autonomously in all aspects of creation offers an opportunity to examine higher-level acts of creativity in videogame design, rather than focusing on a single, easily-isolated subtask and leaving the creative direction to human designers, composers and artists. This also allows us to build systems which are more creatively independent, which leads to the development of software which is not just a tool used in the process of creation, but a collaborator actively contributing to it. By building software which can act autonomously, we force ourselves as researchers to tackle problems that emerge at the intersections between two creative acts (such as the interplay between level design and ruleset design in a videogame). This can highlight problems which may not have been clearly defined had we remained working on small, single-objective generative tasks.

We describe here a piece of software which can autonomously design games by generating several pieces of content simultaneously, including level designs, content layouts, rules, and audio and visual theming. It does this using a particular variation of computational evolution so that the pieces of content complement each other. We will give examples of games produced by the system, and a detailed account of how they were created. We will provide a summary of some experiments performed to evaluate our software, and use it to highlight the difficulties in assessing systems which evolve entire game designs.

This paper is organised as follows: in section II we summarise the process of computational evolution, and the specific variation of it that we have used in our work, namely cooperative coevolution. In section III we introduce ANGELINA, our autonomous game design software, and give a description of the methodology we have used in designing the various iterations of the software. In sections IV and V we describe two iterations of the software, including details of the evolutionary system and additional processes at work within the software. Some of the work described in these sections have previously been described in [9] and [11] respectively. In section VI we give descriptions of several evaluative experiments conducted on versions of ANGELINA, and discuss the results and their meaning for future evaluations of autonomous game design systems. In section VII we place our work on ANGELINA in the context of other research which touches on automating game design, and finally in section IX we summarise our work and provide some conclusions.

II. Background - Cooperative Coevolution

A. Computational Evolution

Computational evolution is a commonly-used search technique for solving combinatorial optimisation problems. Starting with some population (typically randomly generated) of potential solutions to a problem, an evolutionary system will evaluate each in turn using a *fitness function* which provides a metric for sorting solutions. Some fraction of the best solutions will be selected to generate a new population, using biologically-inspired concepts such as crossover (where the

This research was funded in part by EPSRC grant EP/J004049.

features of two solutions are combined to create a *child* solution) and mutation (where features are varied within a single solution to create a new version). This new population is subsequently evaluated, and the process continues until a stopping condition is met.

Computational evolution has been applied to many different problem domains and many variations of it exist, some biologically-inspired, others not. Such techniques have also been applied to procedural content generation for games. A notable example is the game *Galactic Arms Race* [1], which uses evolution to generate new weapons for players dynamically. Here the quality of a solution is evaluated by using player selection as the fitness function. This is called *interactive evolution*, and is commonly used in design tools such as interactive art software [2]. Computational evolution has limitations that often restrict its use in real-time content generation, primarily its incapability of guaranteeing a minimum quality of output in a given timescale. Nevertheless it is often employed in design tools or so-called *offline PCG* [3]. Many variants on computational evolution exist, but typically these variants do not specifically deal with interactions between separate evolutionary systems. Multiobjective evolution, for example, allows for different and competing fitness metrics to be applied to an artefact, but does not allow different parts of the artefact to be evolved separately. An important goal when designing ANGELINA was to build a system with flexibility and a modular design, and for this reason we turned to cooperative coevolution as an underlying evolutionary system.

B. Cooperative Coevolution

One variation on standard computational evolution is to employ a *coevolutionary* approach to solving a problem. In nature, coevolution occurs when changes in organisms cause evolutionary changes in other organisms that share the same environment. For many problem domains, particularly multi-agent systems, modelling evolutionary problems as coevolutionary ones can improve the quality of evolved solutions [4]. Typically this is modelled competitively, where agents attempt to evolve the best possible behaviours for their environment, independent of other agents. As the behaviour of one agent changes, it affects the environment the other agents inhabit. This exerts evolutionary pressure on all agents, the nature of which changes rapidly as each generation brings new agent behaviours and interactions. Competitive coevolutionary processes use this additional pressure as another way to guide the development of the agents towards behaviours which work well in many different scenarios, and in the presence of a diverse range of competing behaviours.

Cooperative variations on coevolution can also be effective, and particularly applicable to problem domains where a solution consists of a combination of several distinct and dissimilar components [5]. A cooperative coevolutionary (CCE) system [6] is composed of several distinct evolutionary subsystems, called *species*, which all solve a particular problem. In order to evaluate a member of a given species, however, that member must first be combined with solutions from the other species making up the CCE program. These solutions are synthesised together to create a complete solution to the original higher-level problem the CCE system is trying to solve. This can then be evaluated, and the results fed back to the subsystems to be interpreted as fitness data. For example, work by Potter and De Jong [7] shows an example CCE system solving an optimisation task, and includes a comparison with other evolutionary approaches. The example system solves a function optimisation problem, where a mathematical function which contains several variables has optimal values for those variables set by a CCE system. Each species of the system is responsible for evolving a particular value which parameterises the function, with co-operation being necessary to evolve not just the value of a specific parameter, but how the *set* of parameters combine to optimise the final calculated value of the function.

III. ANGELINA - CCE for Game Design

ANGELINA[1] is a cooperative coevolutionary system which produces complete, playable games. Different versions of ANGELINA have been developed for specific target genres or game technologies. However, the core structure of the system and its employment of CCE has remained the same throughout. This section will describe two versions of ANGELINA and the games they are capable of producing. Different versions of ANGELINA are denoted by subscripts, such as ANGELINA$_1$, ordered chronologically. These sections focus on implementation and structural details. Evaluation and experimental results for the versions of ANGELINA referenced in this paper are given later, in section VI.

A. Core Structure

The development of all versions of ANGELINA involves the same design procedures. First, we select suitable libraries and technologies to form the basis of the system's design space. Using libraries is very useful for the representation of game entities and the inner logic of rendering and organising objects in memory. Not only can it help in focusing the research on higher level design and creativity questions, but it also helps develop a system that is capable of producing games that are attractive, playable and easily distributed. These features have been high priority from the first version of ANGELINA – aside from the desire to share our results with the wider community, automated game design relies on evaluation through play, and choosing technologies that facilitate this is therefore crucial. We give examples of technologies and platforms used by various major versions of ANGELINA in the subsequent case study sections.

After choosing base libraries that will support the core system, abstractions must be chosen for the concepts the system will be dealing with, such as level designs or player objectives. The level of detail and structure of an abstraction for a given game feature affects the size of the generative space, impacting both the difficulty of the generation process and the potential for novelty. The choice of abstraction in commercial procedural content generators is a defining feature of the generator, and many abstractions have become

[1] A Novel Game-Evolving Lab-rat I've Named ANGELINA.

very common in normal game development. These include representing levels as arrays of integers (adopted by the middleware level design tool *Tiled* [8]) or grammatical systems for representing rules. However, the choice of abstraction is particularly important for automated game design systems, and more so for those employing CCE, because the individual strands of design will exchange information during the design process. What may be an acceptable level of abstraction for one type of generator may not provide sufficient levels of detail for other generators to make decisions about their design. For instance, a generator of level designs may only distinguish between solid and empty tiles, while a generator dealing with enemy behaviour may wish to know if some tiles have special effects or constraints.

Once abstractions are well-defined, we can implement the core CCE system, with appropriate generators and evaluation functions. The CCE system may or may not appeal to the chosen game libraries defined in the first step above in order to evaluate its fitness functions. Alternatively, the system may rely on predefined abstractions of the game in order to evaluate interactions. Some versions of ANGELINA use abstracted simulations of gameplay in order to evaluate interactions between certain species. Others use direct execution of game code in order to assess fitness. While the specifics for any given automated game design system will depend on the technical details of the target domain, we describe the differences in the structure of different versions of ANGELINA in the subsequent case study sections.

The core CCE system ultimately defers to some kind of finishing module that compiles the internal representation of a game into a finished executable or source bundle. More complex approaches to automated game generation may, in theory, not require this step if they directly modify code as part of their design process, since the final evolved system is already publishable. However, most systems – particularly CCE systems, which rely on abstractions for the evaluation and manipulation of genotypes – will have this finishing step as a final way to translate out of abstractions and into a playable result. ANGELINA uses inline code replacement, where code is inserted into pre-existing template programs to implement key functionality, as the means of producing runnable output. This varies slightly according to platform as described in the following sections.

IV. Case Study: Arcade Game Design

The first version of ANGELINA [9], produces simple arcade games using a prototype CCE system. The system's domain is strongly influenced by work in [10] which attempted to design game rulesets using neural networks as a method for evaluation. ANGELINA$_1$ is an expanded system working in the same domain, developing level designs and object layouts (that is, the placement of game content within a level) simultaneously with the rulesets.

A. Platform & Compilation

ANGELINA$_1$ is written in Java, using internal abstractions to represent a grammar of game components. The evolutionary system relies upon the Monkey HTML5 engine to output its games – several template files have been created to describe a skeleton game in the Monkey engine, with markers indicating where information such as the level layout or the rules of the game should be inserted. At the end of an evolutionary cycle, ANGELINA$_1$ automatically converts the game components in the resulting game into boilerplate code that can be inserted into the game templates and compiled.

B. Abstractions

The CCE core of ANGELINA$_1$ has three species in it; ruleset design (RS_1), level design (LV_1) and layout design (LY_1). RS_1 uses a grammar of game rules based on collision, very similar in nature to those described in [10]. We use a single rule, consisting of five parts:

```
Type1 + Type2 : Effect1 + Effect2 + Score
```

This rule states that given an object of type `Type1` and an object of type `Type2` colliding, apply `Effect1` to the first object, `Effect2` to the second object, and add `Score` to the global high score. Object types are selected from entities in the game world, including walls, the player object, and several coloured entities, while Effects are hardcoded game concepts such as random teleportation and object death.

LV_1 uses a two-dimensional array of integers to represent a tile-based world of passable and impassable blocks. LY_1 consists of a list of game objects along with information such as their starting location and control type. There are four basic types of object - one player object, which is grey and responds to keyboard input, and three NPC objects which have control schemes attached such as *static* (not moving at all) and *rotational* (turns right whenever it cannot move forwards).

Figure 1 is a screenshot from a game output by ANGELINA$_1$. The black tiles are impassable and provide an architecture for the map.

C. Internal CCE System

Generators for all three species use random initialisation to produce a starting population, with random integer arrays and selections from grammars used to generate levels, layouts and candidate rulesets. For the former two – level designs and the layout of entities – random integer arrays that span the height and width of the level space is enough to randomly initialise a species. The latter, rulesets, uses a grammar based on the abstract rule template described in the previous subsection. `Type` objects are chosen from the player, the walls, and the three non-player entity types, while effects are one of teleportation, death, or no action. Much of the grammar is inspired by the domain described in [10]. Evaluation of each species can be thought of as being undertaken in two parts: an *internal* evaluation that uses objective evaluations to assess the fitness of a given member of a population, and an *external* evaluation that considers a population member in the wider game context.

Internal evaluations focus on globally optimal features of a particular species. In the case of ANGELINA$_1$, the three

systems select for certain features regardless of the rest of the game's design. For example, RS_1 evaluates potential rulesets according to whether they were *directed* (in that at least one rule raises the player's score) and *challenging* (at least one rule harms the player or impedes progress). LV_1 penalises level designs that contain fenced off areas or have too high a proportion of the level set as impassable. Proportions such as this could be varied by a user prior to execution of ANGELINA$_1$, although in practice we tested values experimentally and settled on ones which produced a good variety of levels that were neither too dense nor too sparse to find interesting designs within.

Most fitness data comes from external evaluations, which require both the player and the game itself to be simulated. This is achieved using a basic reimplementation of the game engine's approximate functionality in Java. Simulating the game engine rather than running the game directly is a question of technical feasibility primarily: because ANGELINA$_1$ is written in Java, while the output engine is HTML5, it is difficult to start and control instances of the game itself from within Java. Instead of starting and closing many hundreds of instances of different game configurations, we chose instead to implement a cut-down game engine inside ANGELINA$_1$ in Java. While not an exact replica, this allows for comparisons to be made between rulesets and layouts quite easily. The simulation can be parameterised with abstractions from RS_1, LV_1 and LY_1 and can then be run any number of times, gathering information on the play session such as the number of rules fired, average level coverage, and other useful metrics about the interplay of the various game components.

In order to properly evaluate the game, we attach the player character to an AI controller which attempts to play the game with a basic understanding of the ruleset. We use three variants on AI controllers in ANGELINA$_1$: static controllers (that do nothing), random controllers (that rapidly cycle between random inputs) and three intelligent controllers with varying levels of risk aversion. The intelligent controllers attempt to seek out collisions that improve their score, and avoid collisions that would diminish their score or kill the player. The level of understanding is basic – if there are side effects to actions, such as one rule removing objects which cause score gain in another rule, the player controller will not be able to seek them out – but it is enough to gain basic metrics for the quality of a game's design. In particular, it is able to deduce whether the game is unwinnable, or unfair, by comparing the performance of intelligent controllers to random or static ones. Additionally, the intelligent player controllers can also be given a risk factor that dictates how close they are willing to come to danger before moving to avoid it. This allows for a basic comparison of risk-averse players to more daring ones, which can give useful insight into the nature of the challenge presented by the game designs. The three intelligent controllers represent high, medium and low risk aversion, hand-tuned to represent a reasonable spread of play style.

A game design is played out three times by each of the different player controllers listed above, to avoid a single anomalous playout skewing the fitness data. We then feed data back to the individual components that make up the game design. The playouts of each controller record whether the controller died, how fast they died, what their final score was, which areas of the map they explored, and which rules were activated in the ruleset. This data gives lots of useful information about how the three different species interact with one another. For example, we can detect whether there are rules which are never activated, and therefore not appropriate for the level design. We can look at how the rules affect the regions of the level that are explored, and whether the player is encouraged to explore or whether they can gain a high score simply by staying in one place.

In most cases, this data is only applicable to one species at a time. This is because in order to evaluate a particular member of a population, it is compiled into a game for evaluation by combining it with the highest-fitness members of the population of the other species. This means that a single game simulation only provides meaningful evaluation feedback for one species at a time. Once the game has been evaluated, it is used to perform an external evaluation on the component being evaluated, which is summed with the internal fitness to provide a final fitness score. This score is used to sort the population prior to the production of a new generation. For an example of an external fitness function, LV_1 evaluates level designs using an equally-weighted sum of three metrics: the percentage of the level the player controller covers during play, the percentage of the level the NPC entities cover during play, and the number of single-tile *bottlenecks* in the game level as a proportion of a target number of bottlenecks set before evolution began. A bottleneck is a single tile which partitions a level into two regions, which are only accessible by passing through that tile.

In order to produce a new generation of level designs, the integer arrays of two parents are combined using single-point crossover. Mutation of a level design is done by randomly selecting between 1 and 10 elements in the array describing the level, and then changing the integer in their array entry from solid to empty or vice versa. For rulesets and layouts, the reproduction procedure is less straightforward; crossover is performed as if the solutions were represented as lists, so rulesets are split at the granularity of rules, and layouts at the granularity of single game objects. Mutation affects the solutions at a finer degree of detail, able to alter individual coordinates on game entities in layouts, or individual types and effects in a single rule. All species within the CCE use elitist selection to produce subsequent generations – internal and external fitness scores are summed together and the highest-scoring individuals are selected to create the next generation. We retain these highest-scoring individuals as part of the new generation also. We found that CCE systems tend to fluctuate wildly in their initial generations before co-operative trends have emerged in the designs, which can result in good parents producing much less fit offspring. By retaining the parents of each generation, we give the system a chance to recover from drops in fitness, which gives the species more time to develop traits that co-operate with the other species.

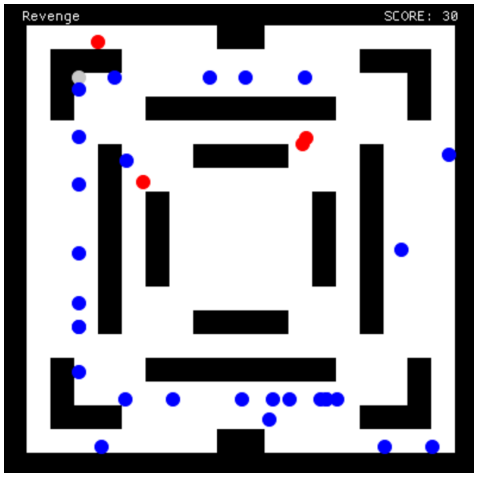

Fig. 1. A screenshot from *Revenge*, a game co-developed by $ANGELINA_1$ in HTML5 using a human-designed map, as part of an experiment in co-design. Playable at http://www.gamesbyangelina.org/games

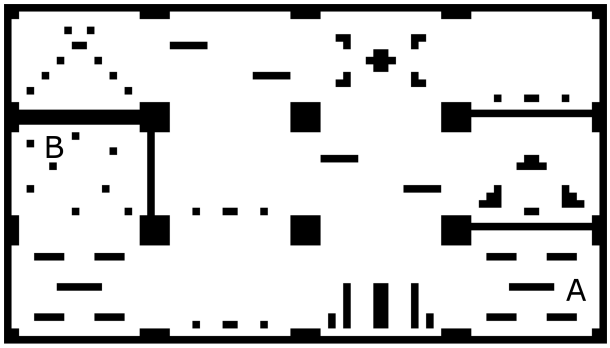

Fig. 2. A level from *Space Station Invaders*. There are two powerups in the level, the first marked *A* and the second marked *B*. Powerup A is designed only to give the player the ability to reach Powerup B by reducing gravity so that their jump is exaggerated. However, the rest of the level is not accessible until Powerup B is reached, which increases their base jump height.

V. Case Study: Metroidvania Design

The second version of ANGELINA [11] demonstrates that CCE-based approaches to game design can be expanded to different genres of game. The system's domain is a subgenre of platform games colloquially referred to as *Metroidvania*. Such games are characterised by large game worlds which the player slowly explores by gaining new abilities that overcome obstacles to progression. This sense of progression is a key identifying feature of Metroidvania games, and easy to quantify formally, therefore making it a good subject for evaluation.

A. Platform & Compilation

$ANGELINA_2$ is written in Java, using abstractions similar to $ANGELINA_1$ to represent game concepts. $ANGELINA_2$'s target platform is Flash, partly because of its ease of distribution, but also because of a proliferation of simple game libraries being released for the platform. $ANGELINA_2$ uses Flixel[2], one of the most widely used libraries for small game development, particularly of two-dimensional games with a platforming component. As with $ANGELINA_1$, template code is used with code sections that are dynamically replaced to compile the final version of the game.

B. Abstractions

The CCE core of $ANGELINA_2$ has three species in it; powerup design (PW_2), level design (LV_2) and layout design (LY_2). PW_2 is the mechanical core of $ANGELINA_2$ and a replacement for the rules-based approach in $ANGELINA_1$. It evolves designs for powerups, which are collectable items that change the game rules in some way. They are formulated by combining a specific variable target, and a value to set the variable to. Three variable targets are available in the abstraction: `jump_height` which controls the velocity applied to the player when the jump button was pressed, `gravity` which sets the downward acceleration for all objects in the game, and `collision_index` defines which tiles are solid and which are passable. While the number of variables is small, the space of possible values to set the variables to is large, making it an interesting problem to optimise for, particularly when considered within the context of a larger CCE system.

LV_2, much like LV_1, uses a two-dimensional array of integers to represent the world in terms of tiles. Unlike LV_1, however, the world is not composed of simply passable and impassable tiles. This time, the integer value of the tile indicates its role in the game logic: a tile denoted by a number i is solid only if its value is greater than the value of `collision_index`. This separates the integer array into solid and non-solid blocks as before, however by modifiying `collision_index` we can now alter which tiles are solid at runtime. Additionally, levels in $ANGELINA_2$ are not generated by placing individual tiles as in LV_1. Instead, pre-made level segments are selected and placed in a grid arrangement, inspired by the approach in [12]. Segments can be laid on top of one another to increase variation in level designs and control level flow more precisely, by blocking off certain directions. Figure 2 shows a level made by the system, with the seams between the segments clearly visible.

LY_2 is similar to LY_1 in structure. It contains a unique object representing the player's start location, and another unique object representing the exit. It also contains a list of enemy objects, along with information such as the enemy type and starting location, and the location of any powerup objects as well. Enemy type refers to one of a number of enemy archetypes, also defined within the LY_2 species. An enemy archetype includes a selection of one of a number of preset behaviours for movement and for reacting to the player's presence, such as a left-right patrol, flying, or pouncing.

C. Internal CCE System

Generators for the species in $ANGELINA_2$ all use random initialisation. Powerup designs have their properties chosen at random, as well as the locations and configuration of the enemies, player and exit objects. Levels are generated by picking level segments at random, including overlays to randomly

[2]http://www.flixel.org

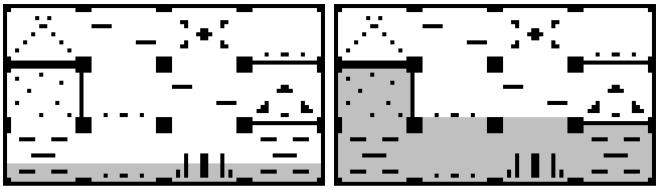

Fig. 3. Two shadings of Figure 2 showing reachable areas both at the level start (left) and after collecting the first powerup (right).

block off paths through the level. As with ANGELINA$_1$, evaluation consists of two distinct steps – internal and external.

Internal evaluation focuses on globally optimal features and avoiding objective failure conditions. In particular, PW_2's internal evaluation assesses sets of powerups to ensure they are unlikely to provide overly similar or contradictory effects when collected. LY_2's internal evaluation selects for enemy placements which distribute evenly throughout the level (regardless of the topology) as well as those which maximise the distance between player start and exit.

External evaluation considers progression and reachability, which we identified as key features in the Metroidvania subgenre we were targeting. We developed a simple simulation of the target game environment that models the player's ability to move and jump around a level. The simulation can be parameterised with values for things like the player's jump height and other features of the game that are affected by powerups, so that the collection of powerups can be simulated as well and we can model player progression through a level. An external evaluation of a game calculates the reachable area from the starting position and then iteratively considers, for each accessible powerup, which new areas are now accessible and whether they recursively would allow for further exploration. Figure 3 shows the reachable areas of the level in Figure 2 in stages, as the player collects more powerful powerups.

Assessing sequences of powerups that can be collected in a given level allows for a rough calculation of the possible routes through the game to the exit. Because it does not directly execute the game engine and instead uses an approximation of the jumping mechanics, the calculation is not guaranteed to find all paths through the level, such as those that might require careful platforming or the abuse of unforeseen game bugs. It also limits the long-term usefulness, as the simulation must be updated to keep itself in line with the current version of the game engine itself. For the purposes of ANGELINA$_2$, this is not a problem – the world is tile-based at a fairly large granularity and there are no complex reachability mechanics other than locked regions, which is coded into the simulation. For more complex games, a simulation-based approach would either need a lot of detail, or be abandoned in favour of direct simulation of play within the game engine.

The calculation of routes from start to exit enables the evaluation of various aspects of the game's design. LV_2's external evaluation, for example, includes assessing how many paths exist and whether they maximise the sense of progression (that is, gradual increases in the accessible space, with the exit accessible at the last stage). PW_2's external measure is also affected by the proportion of the game map that each powerup gives access to, which ensures that it is neither too small (so as to make it insignificant) nor too large (so as to make the remainder of the game trivial).

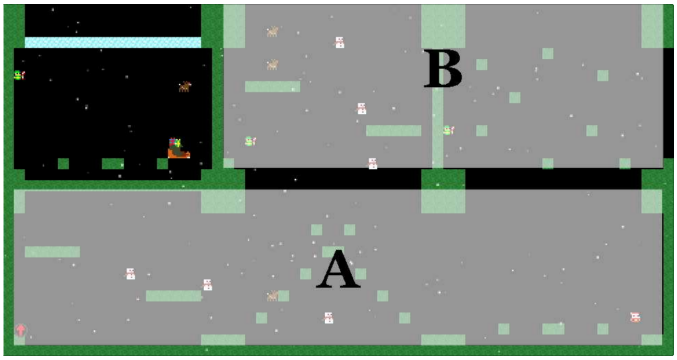

Fig. 4. A screenshot showing two reachable zones in a Metroidvania game created by ANGELINA$_2$. Region A is the player's initially accessible area. Region B is inaccessible until the player finds an item in Region A that changes their abilities.

Unlike ANGELINA$_1$, the ruleset of the games developed by ANGELINA$_2$ are more static, with only the powerups providing variation in the kind of gameplay on offer. There are no player personality models used here as we did with ANGELINA$_1$, as we are primarily interested in the properties the overall game has, and the kind of experience provided by the critical path through the level. In order to provide as detailed a picture as possible of the game world and the routes that exist through it, the simulation attempts to find every possible area the player can access. This is to ensure that no shortcuts are missed by the evaluation function.

After evaluation, a new generation of each species is produced. Crossover of level designs is a mix of one-point crossover and a column- or row-wise swap, wherein a child is created from two parent levels by randomly selecting columns or rows from each parent. Powerup sets are recombined by one-point crossover which affects the list of powerups but does not change the properties of the individuals. Mutations, meanwhile, affect powerups at the individual feature level. Layouts similarly use one-point crossover on the list of layout objects, with mutation affecting the specific features of the start and exit locations, enemy archetypes, and individual enemy placements. As with ANGELINA$_1$, we sum internal and external fitness and select the highest ranking members of the population. Parents are retained along with their children in the next generation, also as in ANGELINA$_1$.

VI. Evaluating Autonomous Game Designers

The evaluation of procedural content generators is not something for which established procedures exist, and remains a point of discussion among researchers in the area [13]. In this section, we describe several evaluations undertaken throughout the course of ANGELINA's development, representing a mixed approach to assessment, in order to show a range of different techniques and highlight the difficulty of this task.

To the best of our knowledge, there is no reliable, tried and tested, default model for evaluating the quality of games

or the creativity of the system that produces them, nor is there a reliable way to measure how people perceive such a system in terms of its creativity or lack thereof. Evaluation methods generally struggle both to isolate decisions made by the software, and to meaningfully assign values to aspects of a game. Subjective evaluations are prone to bias, but the demands of evaluating for creativity prevent us from easily controlling it. If we choose to ignore the games output by the system and only consider the system itself, we run the risk of missing out on the bigger picture.

Composite evaluations which mix in multiple types of evaluation, such as those we employ here, may help to build a foundation for evaluating similar systems. Some artefacts are praised by critics, others by creatives working in the same domain, others are praised by the consumers of the medium, such as players themselves. It may be that by focusing our evaluations on specific groups within the games community, we are able to find more conclusive evaluations for certain aspects of the game.

Creating a unified language with which to describe games made by systems like ANGELINA may also help strengthen the research in the area. While many attempts have been made in the past to propose generic description languages for videogames [14], these often suffered from unnecessary amounts of detail and verbosity, and were unwieldy for applications to autonomous design systems. A concerted effort has recently been made to produce a new, more concise description language called VGDL [15] which may provide a more usable base from which researchers can build autonomous game design systems. This may allow, for instance, comparative studies to be performed where different systems produce easily distinguishable videogames, but are directly comparable because of the shared basic grammar of games that they work from. However, intermediate representation languages such as VGDL potentially harm the creative autonomy of the system by loading it with preconceptions for what a game is, so we are cautious towards their use for the time being.

The question of evaluation for now remains an open problem. We can show positive results in small ways throughout ANGELINA's development, from the analysis of its evolutionary performance in a difficult design space, through to the qualitative, subjective feedback from players of ANGELINA$_2$'s games. So far, though, no approach we have taken has seemed universally applicable or reliable, and we will continue to investigate new ways to evaluate future versions of the software.

A. Sampling The Design Space

Assessing the ratio of playable to unplayable games within the design space is a good indicator of the difficulty of the problem that the CCE system is trying to solve. We generated a set of 100 random game designs, with no evolution, using the abstractions ANGELINA$_1$ employs, and examined each. A game was marked as *Playable* if it had a consistent map and layout (by consistent we mean that they do not overlap conflicting elements, like placing the player in a wall), and a goal-oriented ruleset with a way to gain score, some way of avoiding death if death is present, or similar objectives. If games were not Playable, we mark down the features it is lacking in each of several classifications. The results of the classification exercise are given in Table I.

Playable	3
Illegal Map/Layout	100
Broken Rules	74
No Goal	69
No Obstacles	54

TABLE I
RESULTS OF A SAMPLING OF ANGELINA$_1$'S DESIGN SPACE.

Of the 100 games tested, none passed all checks for playability. Each generated game was compiled and played by hand for verification, and games which had clear goals and obstacles, in which the player had some purpose or way to affect the system, were deemed playable. All of the games in the sample, however, had conflicting layouts and maps. We ultimately allowed a game to be classified as playable for this experiment if the player character was legally placed, although other game entities may be placed illegally. Many entities were cut off from the rest of the map in islands of space, or embedded into walls.

74 of the games generated had broken rulesets. This included rules that would never be executed, such as a rule for collision between a singleton entity and itself, or rules that made no sense for the current design, such as rules for green entities where the layout did not include any. Further to this, 69 of the games had no way for the player to gain score, or avoid losing score, and 54 of them had no challenge for the player, either through death or loss of score. Often a game was rendered unplayable by failing just one of these checks. However, most of the games tested failed three or more checks. This analysis supports the idea that the combinatorial design space explored by ANGELINA$_1$ is sparsely populated with playable, let alone *good*, games.

B. Assessing Fitness Gain

A common way to evaluate evolutionary systems is to consider the change in fitness of a population on a standard run of the system. We hope to show that the fitness function and design of the system encourages gradual increase in fitness to show that the CCE system is functioning as we would expect, and then subsequently show that higher fitness correlates to higher quality games, which we attempt in a later evaluation below. Figure 5 shows fitness plotted over time for a normal execution of ANGELINA$_2$. We can see that fitness increases and plateaus as we would expect, although the early phase of the system shows spiking in the fitness, at one point spiking to higher than the final fitness. In non-CCE systems, we might expect the fitness to increase monotonically. However, because the fitness in a CCE system is the result of all three species and their ability to co-operate with one another, we often find that a single species finds a solution which is highly internally fit, but has low co-operation. This can result in temporary

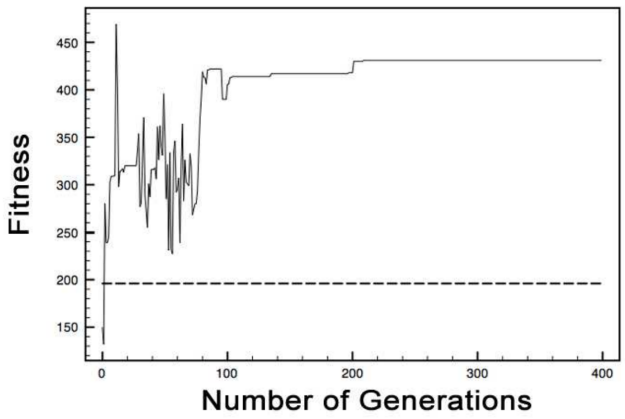

Fig. 5. The fitness of a population in ANGELINA$_2$ throughout a standard execution. The dotted line shows the maximum fitness from a sample of randomly generated games, the size of which is equal to the total number of games evaluated across 400 generations of a standard ANGELINA$_2$ execution.

rises in fitness which drop off on the next generation as that species abandons the outlier in favour of a solution which is less internally fit but co-operates better.

By analysing the performance of ANGELINA both in execution and in output, we can show that the design space is sparsely populated with good games, and we can give support for the idea that CCE performs well for the multifaceted task of game design by showing the fitness gain over time, or by simply comparing the system's output with a less cooperative version of itself. However, this may not be the most effective way of evaluating such a system. This analytical approach leaves us with many questions about the quality of the system's output. Focusing on the performance of the evolutionary system alone gives us no information about whether the games themselves are worth playing, for example. Without some indication that the evolutionary system is maximising features of a game that people find interesting or enjoyable, the analysis doesn't help us evaluate our system very well. Analysing the performance of the CCE system only shows us one side of ANGELINA's performance.

C. Comparing Cooperation

By disabling the cooperative nature of CCE, and only allowing the individual species to evaluate themselves in isolation without the knowledge of a full game design, we can show how ANGELINA functions without cooperation and compare this with its performance normally. Figure 6 shows a game designed by ANGELINA$_1$ without any cooperative features active. The game's rules challenge the player to reach a red object. Touching the red object kills the player, ending the game but gaining score. We see that each component complies with its own fitness constraints – the rulesets approximately provide an objective to the player, for instance, and the map maximises the maze-like aspect of the evaluation function. Despite this, the resulting game is unplayable, because the map and layout conflict with each other by overlaying walls with entities. This means characters are unable to move or be activated by the rules that govern them. The ruleset also

```
<scorelimit>44</scorelimit>
<timelimit>72</timelimit>
<redmovement>CLOCKWISE</redmovement>
<greenmovement>RANDLONG</greenmovement>
<bluemovement>RANDLONG</bluemovement>
<rules>
  <rule>PLAYER,RED,DEATH,DEATH,1</rule>
  <rule>GREEN,BLUE,TELEPORT,DEATH,0</rule>
  <rule>OBSTACLE,RED,NOTHING,TELEPORT,0</rule>
</rules>
```

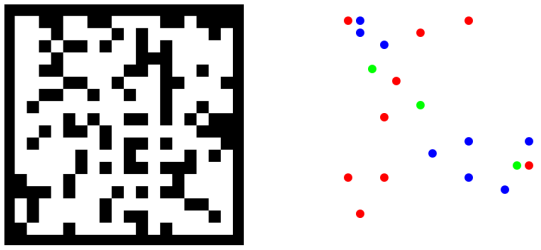

Fig. 6. A game designed by ANGELINA$_1$ without co-operation between the individual species. Note that although the components are individually fit, the components do not relate to each other and the resulting game is unplayable.

```
<scorelimit>24</scorelimit>
<timelimit>96</timelimit>
<redmovement>STATIC</redmovement>
<greenmovement>RANDLONG</greenmovement>
<bluemovement>RANDSHORT</bluemovement>
<rules>
  <rule>OBSTACLE,PLAYER,NOTHING,DEATH,0</rule>
  <rule>PLAYER,RED,NOTHING,TELEPORT,1</rule>
  <rule>RED,OBSTACLE,TELEPORT,NOTHING,0</rule>
</rules>
```

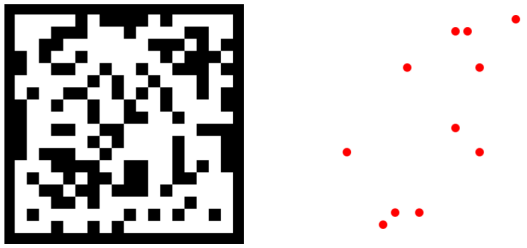

Fig. 7. A game designed by ANGELINA$_1$ with full cooperation between the species. This game defines a quasi-'Steady Hand Game' and was outside the inspiring set for the design space.

includes rules which cannot fire (because of the nature of the layout). These are all features of the game that would be evident were it to be played and evaluated as a whole.

By contrast, Figure 7 shows the result of the system operating with all its co-operative features enabled. The resulting game has a sensible ruleset, no conflict between map and layout, and is generally playable and interesting. The player must collect static red objects by touching them, but must not touch the sides of walls. When a red object is collected, the player is randomly teleported to somewhere else on the map and must quickly make sure they do not hit a wall. The emphasis on reaction times and unpredictability makes

	Best Rank	Middle Rank	Worst Rank
HighFitness	19	9	11
MedFitness	9	15	15
LowFitness	11	15	13

TABLE II
TABLE SHOWING THE RANKINGS BY PLAYERS OF THREE GAMES OF DIFFERENT FITNESS, GENERATED BY ANGELINA$_2$.

the game an interesting one, and not one we anticipated from the initial design space.

D. Evaluation with Players

Games are designed to be played by people, and evaluations should reflect this. By focusing on the quality of the system's output, we can show its performance as a game generator or game designer in its own right, and demonstrate that it produces artefacts of some value. The idea of evaluating a system based on the quality of its output can be seen throughout games research [16], [17] as well as in the field of Computational Creativity [18].

As part of evaluating ANGELINA$_2$, we conducted two surveys with players, in which they were asked to play games made by ANGELINA$_2$ and give feedback on them. We performed an initial pilot study in which 180 players were asked to play the same game and provide a rating between 1 and 5, and asked for qualitative feedback on the games. We noted that players often either cited problems with or praised aspects of the game's design which were not controlled by ANGELINA$_2$ – the control scheme, for example, or the graphical design. We made adjustments to the CCE system to avoid some of the obvious pitfalls the players highlighted, but many issues (such as players evaluating content not affected by ANGELINA$_2$) remained unresolved.

The pilot study was followed up with a more in-depth study in which 35 of the original 180 participants were invited back to play three games made by ANGELINA$_2$ and give feedback on them. The games presented to them had varying levels of fitness, which were played in a random order for each participant. After playing all three, the players were asked to rank the games in order of perceived quality. Our hypothesis was that higher fitness games should result in higher ratings. The results of this survey are shown in Table II.

We found almost no difference between low and medium fitness groups, suggesting players found it hard to differentiate between games at the lower end of the fitness range we sampled. We found that a greater proportion of high fitness games were ranked highest, however: 49% of players compared to 25% ranking low or medium fitness games highest. However, the effect was not significant (chi-squared, p=0.15). We found a very weak and insignificant rank correlation between fitness and player preference, (Kendall's $\tau = 0.11$, $p = 0.17$). In both tests, we were unable to reject the null hypothesis. The results were ultimately inconclusive, even if they may suggest some relationship between fitness and preference exist, that could be investigated with further study.

Studies which target quantitative feedback are difficult to carry out in a situation that has so many variables in, and this situation will only become more difficult as autonomous game designers become broader in scope and able to produce even more varied games in their output. Accurately eliciting an individual's preferences with respect to a set of games is likely only to provide at best a partial order. Players may also have difficulty articulating their preferences, with multiple aspects of a game contributing to their decision in complex ways. There are also complications arising from the combination of fixed design features, such as the control scheme of ANGELINA$_2$'s games which was specified by us at design-time, and design features the system has control over. Many players made evaluations based on aspects of the games which ANGELINA$_2$ had no control over. Shaping and overcoming this limitation requires either a much more complete game design system, or a more carefully designed evaluation that takes into account the specific interactions the player may have with game design features provided by the system designers rather than the system itself. Despite this, it is useful for developers of autonomous design software to continue evaluating systems in this way – our results here, although inconclusive, point the way towards further studies that may be able to provide support for the claim that ANGELINA's understanding of fitness is related to the player's perception of game quality. This can lead into other evaluations that consider qualitative aspects, such as whether players describe the games as playable, or what aspects of the games they find interesting or compelling.

Such evaluations may not be straightforward, however. There is evidence that both positive bias [19][20] and negative bias [21] can affect evaluations of software that acts in creative domains such as videogame design. This can be hard to detect, and the main way of circumventing such bias is usually to perform a Turing Test-style evaluation where the participants in the study are unaware of the creator of the particular artefact they are being asked to evaluate. However, particularly when the assessment of creativity is involved, this approach has received some criticism as it is seen to be covering the problem up rather than treating the existence of bias as part of the research challenge of developing software that works in creative domains [22]. For ANGELINA, one of our objectives is to produce a piece of software that is perceived as being creative. As such, we have no immediate answer to the problem of bias, positive or otherwise, in our study participants, as Turing Test approaches are unlikely suffice.

VII. RELATED WORK

A. Related Work In Game Design

1) The Game-o-Matic: In [23] the authors describe a system called the *Game-o-Matic*, a tool for automatically generating games that portray relationships between concepts defined by a human designer. By defining concepts (such as *police* and *protester*) and the relationships between them (such as *arrests* or *escapes*) the Game-o-Matic can produce playable games in which the relationships between objects are expressed as mechanics that interact between agents in a game world. For instance, given a relationship graph expressing *police arrests protester*, a resulting game may have the player taking the role of a police officer attempting to arrest protester

objects by colliding with them, or controlling a protester who must evade police objects to avoid being arrested.

While the overall aim of the system is to provide an assistive design tool for journalists and other non-designers, the Game-o-Matic is a closely related project to ANGELINA, as it is able to produce playable games with minimal input. Its primary generative task is the creation of game descriptions, which consist of a set of rules and an initial configuration of the game world (including the position of various objects). The rules include an assignment of the player character and an expression representing a goal (such as a score limit).

The Game-o-Matic prepares partial game descriptions based on the initial relationship graph (describing relationships between nouns like policemen and protesters) which it then embellishes using *recipes* to tailor the game design towards a certain type of gameplay. Recipes have two components – a set of preconditions that decide whether or not they can be successfully applied to a partial game design, and a set of modifications that they apply to a game design to tweak the gameplay in a specific way. The Game-o-Matic is designed to allow people without a background in game design or programming to provide basic contextual and cultural information and then co-operate with the system to find a satisfying game design that meets the relationships they expressed initially. It does this very competently – Figure **??** shows a fully-themed and coherent arcade-style game about the food chain and meat consumption, built from a database of game mechanic concepts and a simple four-node concept graph.

The key difference between the Game-o-Matic and our approach with ANGELINA is that we are primarily interested in a system that can operate autonomously in all aspects of game development and design. In particular, we want ANGELINA to be able to generate its own conceptual maps, with an intention of designing a game with a particular theme, and then generate its own mechanical components to include within a game. This is important for the creativity of the system, but also influences our drive towards parallelism over sequential development, since ANGELINA's workflow is not restricted by human interaction at any point and is therefore free to vary the order in which it tackles development tasks.

The Game-o-Matic proceeds in distinct phases which build upon the design generated in the previous phase. This suits the nature of the processes at work within the Game-o-Matic, where design is constrained by initial human input and the primary task is reconciling pre-made components so that a cohesive design can emerge. However, for autonomous game design in the absence of external constraints, we argue that a parallel architecture is preferable as it places no emphasis on particular aspects of the design. This means that progress made in one area of the design can influence the remainder of the design at any time, without explicit reordering of generative steps, as would be necessary with a sequential system.

2) Variations Forever: In [24] the authors present a system for designing simple game configurations using answer set programming (ASP). ASP is proposed as a new method for procedural content generation and a key contribution in [24] is arguing for its strengths, which include simple constraint addition such as requiring that the games produced have a particular feature (e.g. the player moves using a particular control scheme).

Variations Forever is one of only a few games of its kind in the field of procedural mechanic or game design, in that it uses its procedurality as a key part of its appeal as a game. The player is intended to prune the design space through playing particular games to varying degrees of completeness, unlocking new mechanics or design types as they play the game. This blurs the line between the use of procedural generation as a design tool, and its use as an interactive system within the game.

B. Related Work In Mechanic Design

Rules and mechanics are often seen as the core of a game's design, and so although they are only a single species within ANGELINA we can consider systems that produce mechanics and rulesets as related to the work on ANGELINA. In [10] the authors present a system which evolves rulesets for games using a grammar-based domain and an evaluation method that assesses how complex a ruleset is to learn. The domain for this work directly influenced the work described in section IV. However, the idea of generating rulesets and mechanics has since been expanded through ANGELINA by the work described in section V and [25] to reduce the reliance on human-authored grammars and databases of game content.

In [26] the authors describe a system which takes a written input similar to a simplified version of Game-o-Matic's concept graphs, and produces a playable game as a result. The system takes simple inputs consisting of verbs and nouns such as *shoot duck* and then cross-references the words with a database of art and mechanical concepts. The duck example can result in multiple games depending on matches found in the database – for instance, one game may involve playing as a duck and avoiding bullets, while another may have the player controlling a crosshair attempting to shoot a duck. The system operates sequentially, with independent systems working alone, as art and mechanics are developed separate to one another. As with other systems discussed in this section, autonomy is not the primary concern here, and as such there is little freedom for the system to produce novel content or influence its output dependent on what is already generated.

VIII. Discussion and Future Work

The work we have outlined in this paper represents a starting point for a larger investigation into the most effective ways to generate games autonomously, and how best to engineer such systems to maximise the novelty and/or quality of the resulting games. Automated game design is a relatively new field of study, and so many directions remain unexplored by this work – including other genres and the generation of many other aspects of a game's content. Two particular points of future work we are interested in regard the generation of code and researching deeper into issues of design and quality.

A. Code Generation In Game Design

Eschewing intermediate representations in favour of directly using code in the generation of game content is a crucial

step for procedural content generation, and by extension autonomous game design as well. Code is the medium that people work with when designing games, and while using other representations may simplify the design task, they ultimately constrain the design space and enforce too many presumptions on the system which reduce the opportunity for novelty and surprise. Code generation allows procedural generators to work in the same domain as people do, and allows us to be more confident in aiming for systems which are capable of innovating and surprising us.

We have already published experimental work using code generation for arguably the most code-dependent aspect of game design, that of game mechanics [25], and argued for its importance in increasing the perception of creativity in software [27]. We are yet to integrate a code generation process into a CCE designer, however. We aim to integrate code generation techniques at many points in the game design process: to achieve mechanical goals (such as those demonstrated in [25]); aesthetic goals (such as the generation of graphics shaders); or to achieve meta-level design goals, such as code generation for the production of generative processes, which themselves become procedural content generators alongside ANGELINA's static systems.

B. High-Level Design, Quality and Diversity

A game designer may have many aims, but a common one will be to produce games that people wish to play, whether for their meaningfulness, their mechanical depth or their entertainment value. However we decide to expand this, future versions of ANGELINA must look to increasing the diversity of games it can produce, and maximising the appeal of its games one way or another. Our recent focus on the development of game mechanics allows us to further explore the possibility that ANGELINA will provide a source of novel mechanical concepts, which will help increase the appeal of games in terms of their mechanical complexity. Another avenue to explore in this regard is increasing the diversity of ANGELINA's output, which we believe we can achieve with an increased focus on automating high-level decision-making.

Currently, ANGELINA is greatly restricted in what it can output based on the core of the CCE system – the species it uses and the output procedures that finalise the game both restrict the genres the system can work in and the nature of the games it produces. ANGELINA$_2$, for instance, can only produce platformers that include enemies and powerups. Even if we greatly vary the fitness functions to alter the configurations they rank highly, ANGELINA$_2$ cannot change the underlying structure of the game it is going to produce. Yet we understand, as described in work such as [28], that the choice of game components and internal structure is key in designing the overall impact and meaning of a game. Future versions of ANGELINA will explore the possibility that the system may be able to reconfigure the components within a CCE system to adjust the choice of game genre or internal game consistency – removing enemies, for instance, to explore a less action-oriented area of the design space.

Each version of ANGELINA also only generates one genre of game. Of course this could easily be circumvented by combining the systems and executing one randomly, however a better understanding of why a designer might choose to make a particular kind of game could help improve the perception of the software.

IX. Conclusion

We have described ANGELINA, an automated game designer which uses cooperative coevolution, a modular evolutionary algorithm. We described in detail the architecture that connects all major versions of ANGELINA and showed how this methodology, combined with the flexible nature of CCE's *species* modules, enabled us to incrementally improve ANGELINA as well as transfer the core design between different game genres and output platforms. The flexibility of CCE not only makes it an appealing way to develop a system that can act independently of human designers, but also how assistive design tools might be built in future that allow people to choose which aspects of game design they wish to take part in, and have the system adapt itself to fill in the remaining aspects of the game's development.

We have presented a number of evaluations of ANGELINA, from different perspectives and showing differing levels of success. We have investigated the performance of evolutionary systems, considered the difficulty of the design task itself, evaluated our games with feedback from players, and analysed the creative processes that ANGELINA undertakes to create a game. Our overall conclusion is that the evaluation of autonomous game designers remains an extremely difficult task, and no single approach provides a good understanding of how a system is performing. Despite this, composite evaluations may offer promise in future work, and an awareness of the different ways in which such systems can be evaluated may help future projects focus on different aspects of their systems.

Game design's multifaceted nature – incorporating intensely creative activities including art and music, with the complexities of system design, and the technical difficulties of implementation and optimisation – make them daunting challenges to approach as an entire process. The modular nature of CCE not only helps us solve this on a technical level, but it offers us opportunities to do more, and frees designers of such systems from the need to provide explicit structure to the design system being built. Rules do not precede level design in ANGELINA, nor do they follow them – they are designed in concert with one another, giving the system more flexibility and requiring less to be fixed in place by a person prior to execution.

The modern story for procedural content generation research is that it provides a solution to the content problem faced by the large-scale games industry of today, or promotes creativity in human designers by providing more flexible, larger scale tools to work with. Autonomously creative game design offers a new angle on this field of research, however – one that presents a plethora of new challenges, as well as possibilities. This paper shows a series of first steps towards building software that can act independently as a creator, and offers ways for others who follow to compare and contrast their own approaches with us. In doing so, we broaden the field of procedural generation beyond the goals of 'more' and 'faster',

and look instead to how software can break new ground and help people achieve similar goals themselves.

ACKNOWLEDGMENTS

We would like to thank Phillipe Pasquier, Alison Pease, Azalea Raad, Adam Smith, Julian Togelius, Joe Martin, Mike Treanor and Tony Veale for many fruitful conversations over the course of this work's development. We also thank Michael Brough, Darren Grey, Ed Key and Jeff Lait for discussions that helped shape this paper. This project has been supported by EPSRC grant EP/L00206X/1.

REFERENCES

[1] E. J. Hastings, R. K. Guha, and K. O. Stanley, "Evolving content in the galactic arms race video game," in *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, 2009.
[2] J. Romero and P. Machado, Eds., *The Art of Artificial Evolution: A Handbook on Evolutionary Art and Music*. Springer Berlin Heidelberg, November 2007.
[3] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne, "Search-based procedural content generation: A taxonomy and survey," *IEEE Trans. Comput. Intellig. and AI in Games*, 2011.
[4] W. D. Hillis, "Co-evolving parasites improve simulated evolution as an optimization procedure," in *Proceedings of the 9th international conference on Self-organizing, Collective, and Cooperative Phenomena in Natural and Artificial Computing Networks*, 1990.
[5] P. Husbands and F. Mill, "Simulated co-evolution as the mechanism for emergent planning and scheduling," in *Proceedings of the 4th International Conference on Genetic Algorithms*, 1991.
[6] M. A. Potter and K. A. De Jong, "Cooperative coevolution: An architecture for evolving coadapted subcomponents," *Evol. Comput.*, vol. 8, no. 1, 2000.
[7] M. A. Potter and K. A. D. Jong, "A cooperative coevolutionary approach to function optimization," in *Proceedings of the International Conference on Evolutionary Computation*, 1994.
[8] T. Lindeijer, "Tiled," http://www.mapeditor.org/.
[9] M. Cook and S. Colton, "Multi-faceted evolution of simple arcade games," in *Proceedings of the IEEE Conference on Computational Intelligence and Games*, 2011.
[10] J. Togelius and J. Schmidhuber, "An experiment in automatic game design," in *Proceedings of the IEEE Conference on Computational Intelligence and Games*, 2008.
[11] M. Cook and S. Colton, "Initial results from co-operative co-evolution for automated platformer design," in *Proceedings of the Applications of Evolutionary Computation*, 2012.
[12] D. Yu, "Spelunky," 2008, http://www.spelunkyworld.com.
[13] G. Smith, "The seven deadly sins of PCG research," http://sokath.com/main/the-seven-deadly-sins-of-pcg-papers-questionable-claims-edition/, 2013, expanded from a panel discussion at the Foundations of Digital Games conference 2013.
[14] N. Love, T. Hinrichs, D. Haley, E. Schkufza, and M. Genesereth, "General game playing: Game description language specification," http://logic.stanford.edu/classes/cs227/2013/index.html, 2008.
[15] T. Schaul, "A video game description language for model-based or interactive learning," in *Proceedings of the IEEE Conference on Computational Intelligence in Games*, 2013.
[16] D. Williams-King, J. Denzinger, J. Aycock, and B. Stephenson, "The gold standard: Automatically generating puzzle game levels," in *AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2012.
[17] E. McDuffee and A. Pantaleev, "Team blockhead wars: Generating FPS weapons in a multiplayer environment," in *Proceedings of the Second Workshop on Procedural Content Generation in Games*, 2013.
[18] G. Ritchie, "Some empirical criteria for attributing creativity to a computer program," *Minds and Machines*, vol. 17, no. 1, 2007.
[19] S. Colton and G. A. Wiggins, "Computational creativity: The final frontier?" in *Proceedings of the European Conference on AI*, 2012.
[20] M. Cook and S. Colton, "Ludus ex machina: Building a 3d game designer that competes alongside humans," in *Proceedings of the 5th International Conference on Computational Creativity*, 2014.
[21] D. Moffat and M. Kelly, "An investigation into people's bias against computational creativity in music composition," in *Proceedings of Third Joint Workshop on Computational Creativity*, 2006.
[22] A. Pease and S. Colton, "On impact and evaluation in computational creativity: A discussion of the Turing test and an alternative proposal," in *Proceedings of the AISB symposium on AI and Philosophy*, 2011.
[23] M. Treanor, B. Blackford, M. Mateas, and I. Bogost, "Game-o-matic: Generating videogames that represent ideas," in *Proceedings of the Third Workshop on Procedural Content Generation in Games*, 2012.
[24] A. Smith and M. Mateas, "Variations forever: Flexibly generating rulesets from a sculptable design space of mini-games," in *Proceedings of the IEEE Conference on Computational Intelligence and Games*, 2010.
[25] M. Cook, S. Colton, A. Raad, and J. Gow, "Mechanic miner: Reflection-driven game mechanic discovery and level design," in *Proceedings of 16th European Conference on the Applications of Evolutionary Computation*, 2013.
[26] M. J. Nelson and M. Mateas, "Towards automated game design," in *Proceedings of the 10th Congress of the Italian Association for Artificial Intelligence*, 2007.
[27] M. Cook, S. Colton, and J. Gow, "Nobody's a critic: On the evaluation of creative code generators," in *Proceedings of the Fourth International Conference on Computational Creativity*, 2013.
[28] M. Treanor, "Investigating procedural expression and interpretation in videogames," Ph.D. dissertation, University of California, Santa Cruz, 2013.

Michael Cook gained an MEng. in Computing from Imperial College, London in 2010. His research interests include the automation of game design, applications of computational creativity to videogames, and computational models of meaning in videogames. He is currently working at Goldsmiths College at the University of London as a Research Associate in the Computational Creativity Group, and at Imperial College, London where he is a PhD. student. He maintains a research log at www.gamesbyangelina.org.

Simon Colton is Professor of Computational Creativity at Goldsmiths College, University of London, where he leads the Computational Creativity Group (ccg.doc.gold.ac.uk). He was awarded a PhD. in Artificial Intelligence from the University of Edinburgh in 2001. He has published more than 150 papers on a range of AI topics, as well as application papers ranging from poetry generation to the automated discovery of new mathematical concepts, and philosophy of Computational Creativity. He is well known for his software The Painting Fool, the aim of which is to be taken seriously as a creative artist in its own right one day.

Jeremy Gow received a BSc. in Artificial Intelligence & Mathematics from the University of Edinburgh, UK in 1997, and a PhD. in Informatics from the same institution in 2004.

He is a Lecturer in Games Programming at the Computational Creativity Group, Department of Computing, Goldsmiths, UK. His research interests are the modelling of behaviour and experience in interactive systems, and the use of learning agents in design and creative activities, with a particular interest in play and video game design. He has published over 50 research papers on topics in AI and HCI.